### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

**פיצ'ר מס'1 – "Check Friends":**

פיצ'ר המאפשר בעת לחיצת כפתור לבדוק האם אחד החברים עזב ולהחזיר הודעה מתאימה בנוגע למצב החברים העדכני.

**פיצ'ר מס' 2–"Lottery Friends":**

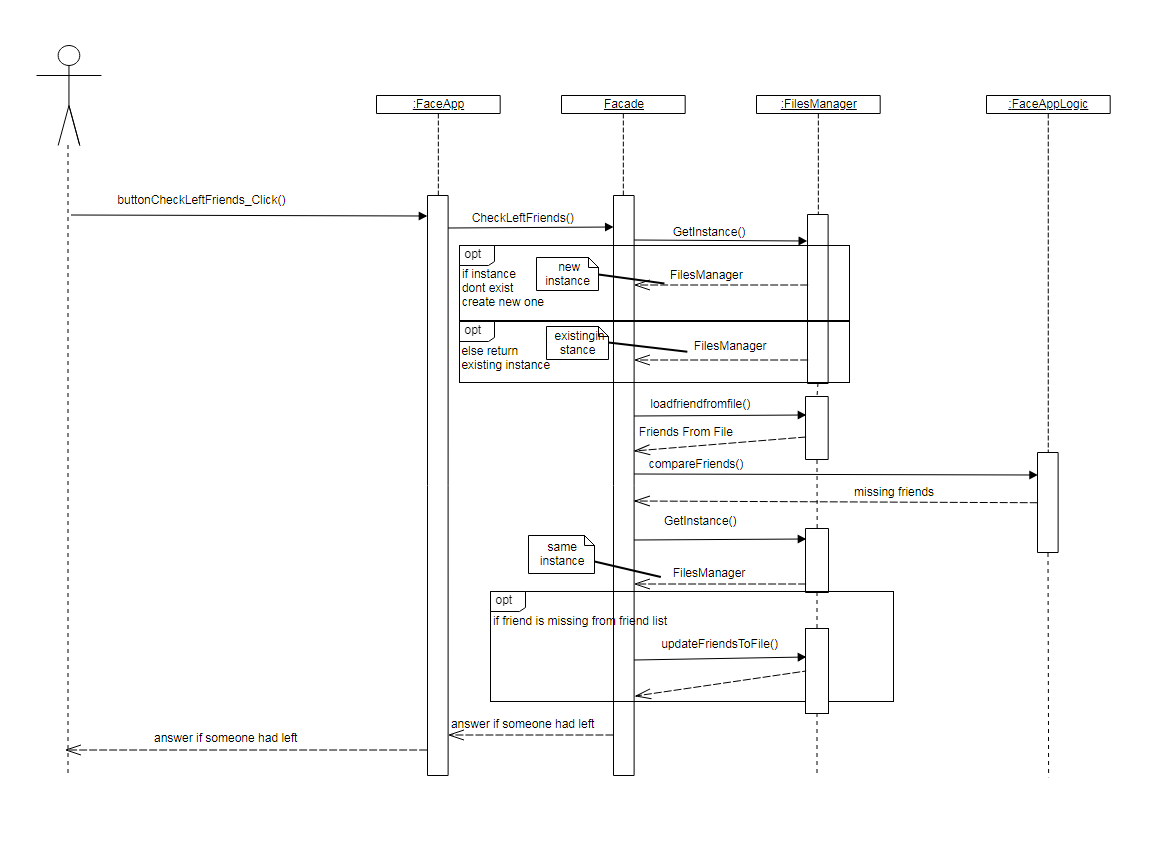
פיצ'ר המאפשר בעת לחיצת על התמונה עם מכונת המזל להגריל חבר מרשימת החברים, בסיום ההגרלה, המשתמש מקבל את פרטיו האישיים של החבר הנבחר.

### תבנית מס' 1 – [Singelton]

* סיבת הבחירה / שימוש בתבנית:
* מחלקה FileManger אחראית על הגישה והעבודה מול קבצי הנתונים של המשתמש באפליקציה ( שמירה וטעינה של פרטי החברים ).
* מכיוון שרצינו שיהיה לכל היותר מופע אחד של המחלקה (מחלקת שירות במהותה) וכן אם בעתיד רכיבים עתידיים יצטרפו למערכת הם יוכלו אם ירצו ביוזמתם לפנות למחלקה (נעשה בה שימוש גם במחלקה FaceApp וגם בFacade-).
* אופן המימוש:

נעשה שימוש במחלקה FilesMnager המציעה שירות סינגלטון Thread Safe ו (fully) Lazy Created – היא זו שלמעשה אחראית על הבאת הInstance. ניתן לראות בקוד במחלקה .FilesManager וכן בשורות 165 ב FaceApp ובשורות 26 ו- 33 ב Facade

* Sequence Diagram



* Class Diagram

### C:\Users\rotem\Desktop\תיקיה חדשה\singelton class diagram.png

### תבנית מס' 2 – [Facade]

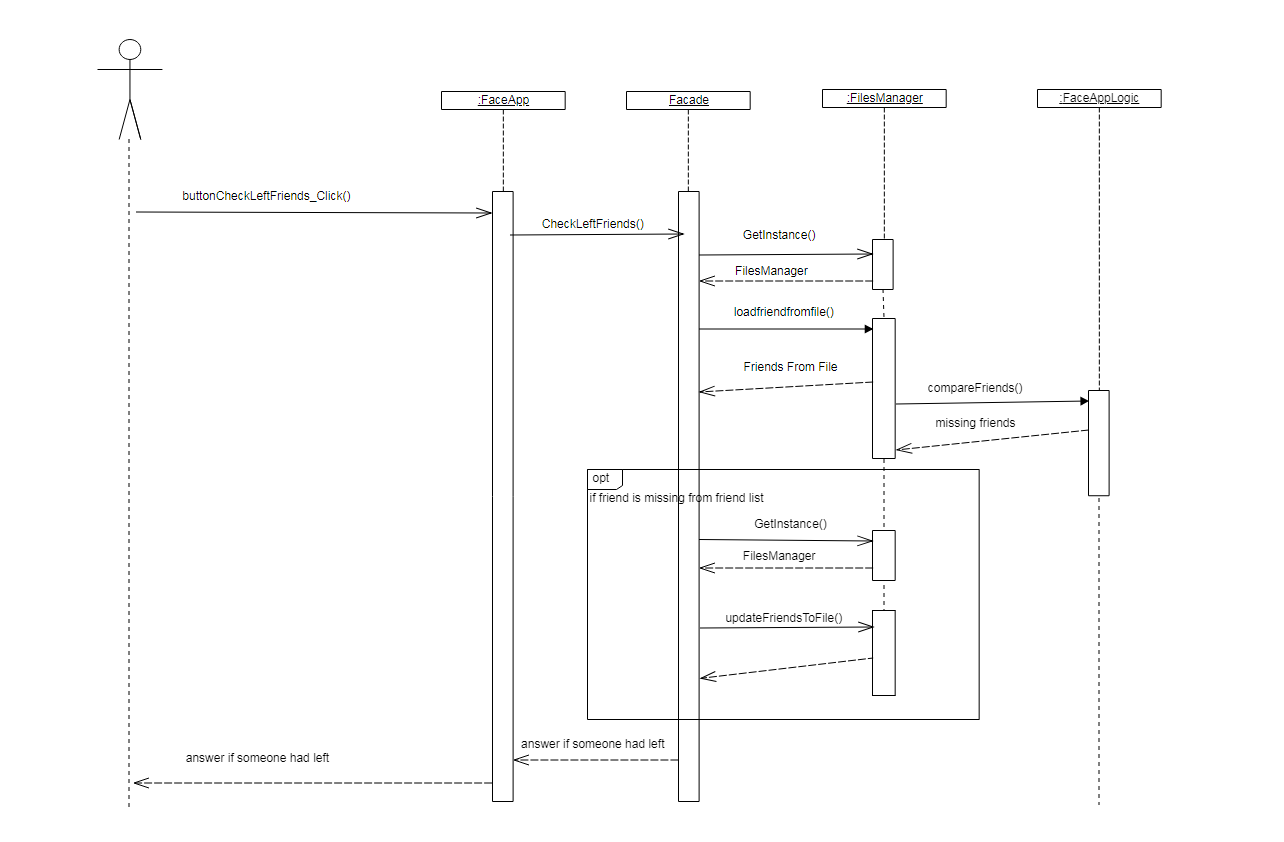
* סיבת הבחירה / שימוש בתבנית:

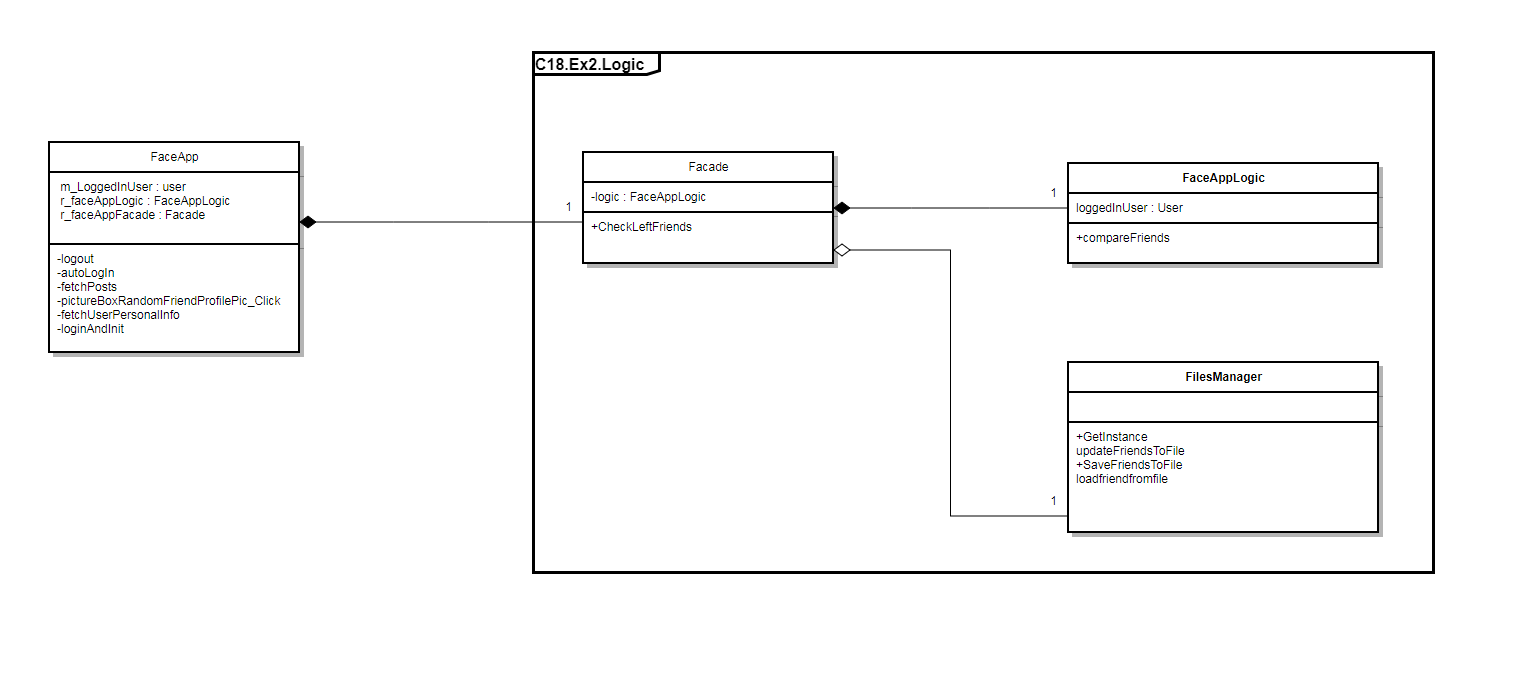
המחלקה Façade נוצרה לשם פישוט הלוגיקה עצם חשיפה של הפעולה המרכזית אותה המשתמש רוצה לבצע כך שכל הפעולות המורכבות נעשות מאחורי הקלעים בצורה אשר לא נראית לעיני המשתמש.  
 כלומר ה Façade שלנו עוזר לנו ליצור ממשק פשוט למערכת גדולה המכילה בתוכה תתי מערכות .

אופן המימוש:

יצירת המחלקה Façade וחשיפת המתודה CheckLeftFriends שורה 20 כך שאנו מסתירים חלק מהמותודת של הלוגיקה באמצעות internal במחלקה FaceAppLogic כדוגמת compareFriends שורה 21 ובמחלקה FilesManager כדוגמת updateFriendsToFile

Loadfriendfromfile שורה 41 ו57 בהתאמה.

* Sequence Diagram
* Class Diagram



### תבנית מס' 3 – [Adapter]

* סיבת הבחירה / שימוש בתבנית:

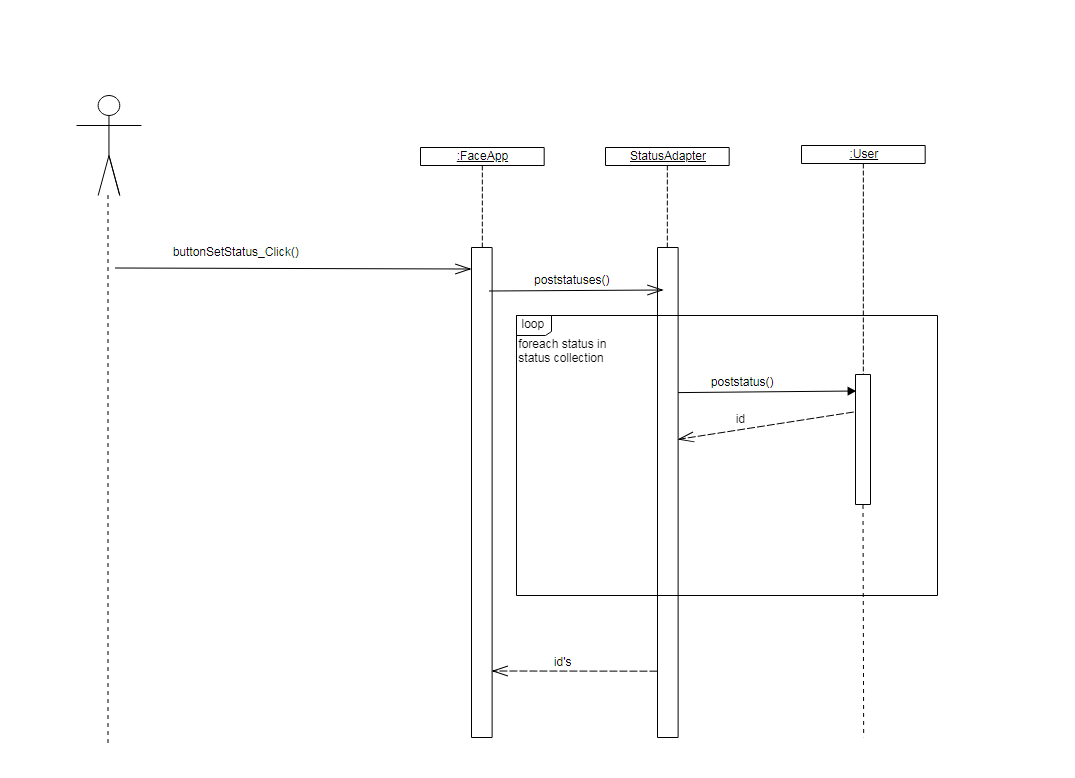
בחרנו את התבנית הזאת כי היה לנו רכיב User שהמערכת שלנו מהונדסת להשתמש בו, אולם התגלה שהממשק שהוא חוסף PostStatus לא מתאים לצרכינו. היות ואין לנו את הקוד של הרכיב User ואין לנו עניין לשנות את הקוד שלו, החלטנו לממש ממשק אחר שיתאים לצרכי המערכת.

* אופן המימוש:

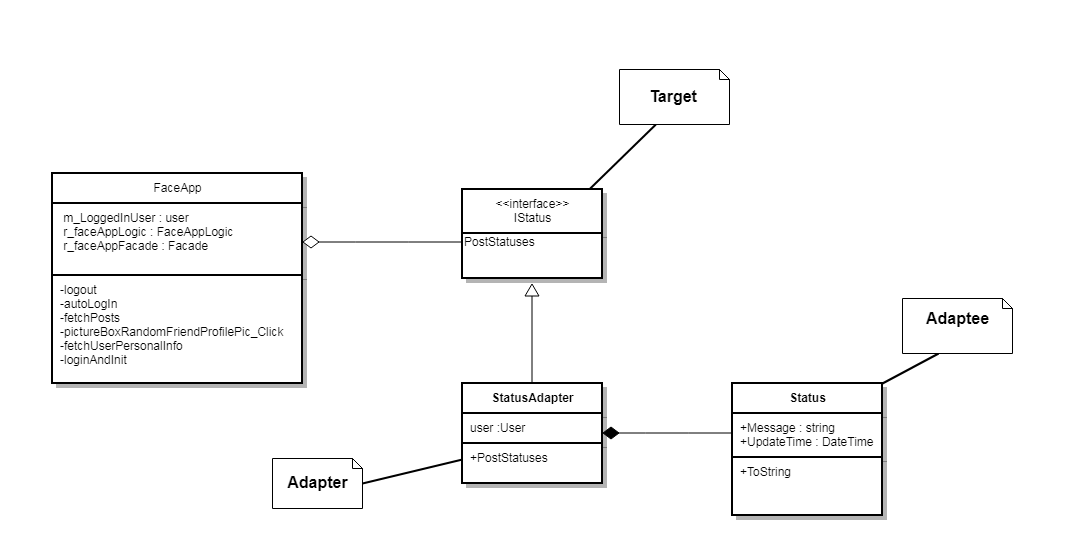
יצרנו את המחלקה StatusAdapter (Adapter) אשר מממשת את ממשק IStatus ה- StatusAdapter יחזיק בקומפוזיצה אובייקט מסוג User (Adaptee) ובכך תתאפשר עבודה בין 2 ה – legacy components ( List<string> ו – string).( PostStatus מצפה לקבל מחרוזת בעוד לנו יש מערך של מחרוזות פה נוצר החוסר בתאימות)

ניתן למצוא בקוד במחלקה StatusAdapter IStatus ו FaceApp בשורות 208-213 .

* Sequence Diagram



* Class Diagram



**עבודה א-סינכרונית:**

ניתן למצוא בקוד שימוש בתכנות א-סינכרוני בפיצ'רים:

* **fetchUserInfo() שורה 129**
* **fetchPosts() שורה 175**
* **fetchUserPersonalInfo() שורה 128**
  + אלו הן הפונקציות האחראיות על הבאת רשימת הפוסטים והמידע אודות המשתמש לListBox-ים כאשר נלחצים כפתורי ה-fetch המתאימים. הפעלתם מThreadים נפרדים משפרת את חוויית המשתמש ע"י מתן האפשרות להמשיך את השימוש באפליקציה תוך כדי הבאת הנתונים.

במקרה הצורך, המימוש עושה שימוש במתודת Invoke של פקדי ה ListBoxים בכדי להפעיל מתודות שמוצגות ב - UI מהThread שבו הListBoxים נוצרו. כמו כן נעשה גם שימוש ב Lambda expressions במימוש זה.

* **autoLogin() שורה 59**
* (כאשר משתמש בחר remember me)
  + סיבת השימוש בדומה לסיבות הקודמות – לאפשר חוויית משתמש רציפה בזמן שהנתונים הבסיסיים של המשתמש נטענים במידה ונבחרה האפשרות remember me.

(כל השורות מתייחסות למחלקה FaceApp )

**עבודה עם DATA BINDING:**

יצרנו userBindingSource שמקושר ל textbox ים שמציג את היומולדת שם פרטי ומידע על החבר שנבחר בהגרלה (פיצ'ר 2 ) כאשר את המידע על החבר הוא מעדכן אם בוצע שינוי ב textbox המתאים ( two way data binding) רק ל מידע על החבר היה ניתן לעדכן כי היה חשוף לו פרופרטי של set בניגוד לאחרים שניתן היה רק למשוך כלומר get